Datalab 实验报告

姓名：任宣宇

学号：2023202303

总分	bitXor	samesign	logtwo	byteSwap	reverse	logicalshift	leftBitCount	float_i2f	floatScale2	float64_f2i	floatPower2
36	1	2	4	4	3	3	4	4	4	3	4

解题报告

亮点

1.float64_f2i

2.leftBitCount

3.logtwo

bitXor

```
int bitXor(int x, int y) {
    return ~(x & y) & ~(~x & ~y);
}
```

思路：纯定义题，根据异或的定义写出逻辑运算式就好 ~~(有一种离散数学的美)~~

samesign

```
int samesign(int x, int y) {
  if (!x && !y)
    return 1;
  if (x && y && !((x >> 31) ^ (y >> 31)))
    return 1;
  return 0;
}
```

思路：这道题比较~~恶心~~有思考量的点在于不能使用或（||）以及对于0的特判

两者均0 返回1 两者均不为0且符号相同（右移31位异或判断）返回1

logtwo

主要实现形式如下

```
int shift = ((v > 0xFFFF) << 4);
result = result | shift;
```

思路：一道藏在level1里的大杀题，主要考察二分法的应用以及对log2的数学掌握程度（bushi

和寻找最左侧的1思路相似，通过比较v与各个标准数字的大小（1，010，1000.....）来确定1的位置

亮点：

一开始没有思路的原因是没有意识到2的整数次方的二进制表述**只有一个1**

byteSwap

~~一开始看错题看成了bitswap，死活做不出来~~

~~以及0x中2个数字才组成了一个byte~~

```
int act_n = n << 3;
int act_m = m << 3;
```

将“第几个位置”转化为第几位

```
int n_byte = (x >> act_n) & 0xFF;
int m_byte = (x >> act_m) & 0xFF;
```

然后将两段要交换的内容提取出来，之后再用或操作拼接就好

reverse

亮点：some magic numbers

- 0x55555555,0xAAAAAAAA
- 0x33333333,0xCCCCCCCC
- 0x0F0F0F0F,0xF0F0F0F0
- 0x00FF00FF,0xFF00FF00

分别代表了相邻2位、4位、8位交换，最后再将前后两段颠倒就实现了所有32位的颠倒

logicalshift

看见本题的第一反应：？ logicalshift和一般的>>有什么区别莫 然后看到测试样例是负数 瞬间老实

所以本题的重点就是如何解决负数移位前置的一大堆1

```
mask = mask & (~(((1 << 31) >> n) << 1));
return (x >> n) & mask;
```

mask初始值为0xFFFFFFFF，(1<<31)>>n的结果左侧均为1（由于1<<31变成了负数）

leftBitCount

思路：本题要求数左起最多位数的1，**且如最高位不是1，结果直接为0**

因此我们考虑将原32位的数逐渐缩短，先确认x的区间在2^(n-1)~2^n之间，然后逐步二分到长度为1位，从而找出精确的连续1个数（这段话非常的抽象，因此上例子）

1.特判：

```
int R_32 = !~x;

x &= R_32 + 0xFFFFFFFF;
```

此处第二行代码感谢李甘（学号：2023202296）同学的思路提醒，由于-1这个数的特殊性，如果用移位法会出现(1<<32)的形式，而这似乎是会报错的(?)

因此用R_32+0xFFFFFFFF来判断R_32是否为1，如R_32为1则会导致溢出，恰好得到我们想要的0x0

2.一般情况

```
int R_16 = !(~(x & 0xFFFF0000) & 0xFFFF0000);

x <<= (R_16 << 4);
```

（1）如果前大于等于16位均为1，R_16=1，x向左移动16位，接下来只需考虑后16位

（2）如果少于16位为1，R_16=0，此处这个！操作非常的灵性，他会使得非0的数字全部变成0，x不移位，说明不用再考虑后16位内容，往后再分别讨论前八位、前四位直到划归为情况（1），即R_8=1或R_4=1.....

3.结果

最后将R_32~R_0用“|”连接即可

float_i2f

IEEE754模版题哈

第一步：x=0特判

第二步：(1<<31)提取符号，确认绝对值

第三步：确认bias，e初始值为最大偏移量158(127+31)

```
while (!(absolute & 0x80000000)) {
    absolute <<= 1;
    e -= 1;
  }
```

第四步：计算尾数frac

```
frac = (absolute & 0x7FFFFFFF) >> 8;
```

第五步：计算rest（即处理进位与否）

floatScale2

主要考察对inf和非规格化/规格化浮点数的转换

如果初始exp已经代表inf 则返回inf

初始非规格化(exp=0)

```
frac = frac << 1;
    if (frac & 0x00800000) {
      exp = 0x00800000;
      frac = frac & 0x007FFFFF;
    }
```

判断乘2后尾数是否可以变为具有前导1的形式，并转化为规格化浮点数

值得注意的是 乘以2之后变成inf的情况

float64_f2i

~~吐槽：刚开始真的很想定义long long类型哈~~

特殊规定：overflow时返回0x80000000 underflow返回0

考虑到int 只有32位的精度问题，我们只需考虑尾数的30位+前导1

```
int frac = 0x40000000 | ((uf2 & 0x000FFFFF) << 10) | (uf1 >> 22);
```

然后根据exp判断是否溢出（上+下）

如果没有溢出：

```
int result = frac >> (30 - exp);
```

最后将符号位用“|”连接即可

floatPower2

首先非常感谢柴老师在DDL之前上课讲了这道题~~(bushi)~~

本题特殊之处在于2^n只会出现1个“1”

1.判断过小：非规格化表示的最小的数，exp=0且frac=000……01，此时值为2^(-149)

2.考虑非规格化浮点数，直接设置exp=0

```
else if (x < -126) {
    exp = 0;
    frac = 1 << (x + 149);
  }
```

3.考虑规格化浮点数，由于是2^n形式，因此尾数必然为0（只保留隐含前导1唯一的“1”）

```
else if (x < 128) {
    exp = x + 127;
    frac = 0;
```

4.考虑溢出，x>128时设置exp为0x7F8